Laboratorul 08.

ATENTIE! Se vor folosi tipuri generice in toate problemele!

Arhiva laborator

Problema 1

Să se definească o clasă **generică** ***ArrayMap*** pentru un dicționar realizat ca o colecție de obiecte ***ArrayMapEntry*** (colecția va fi obiect de tip ***ArrayList***). Clasa ***ArrayMapEntry*** va implementa interfața ***Map.Entry*** și va avea următoarele metode:

```
public class ArrayMapEntry<K, V> implements Map.Entry<K, V> {
private K key;
private V value;
...// constructor
public K getKey();
public V getValue();
public V setValue(V value);
public String toString();
public boolean equals(Object o);
public int hashCode();
}
```

Clasa ***ArrayMap*** va extinde clasa ***AbstractMap*** și va defini metodele:

```
public class ArrayMap<K, V> extends AbstractMap<K, V>{
...
public Set entrySet();
public int size();
public V put(K key, V value);
....
}
```

Pentru testare, se va folosi clasa ***Task1***, pusă la dispoziție în arhiva laboratorului.

Clasa ***ArrayMapEntry*** trebuie să fie clasă internă.

Ce putem spune despre interfața pe care o implementează clasa ***ArrayMapEntry***?

equals() [https://docs.oracle.com/javase/7/docs/api/java/util/Map.Entry.html#equals(java.lang.Object)] și hashCode() [https://docs.oracle.com/javase/7/docs/api/java/util/Map.Entry.html#hashCode()]

Problema 2

Definiți clasa ***Catalog*** care moștenește clasa ***TreeSet***, specificând tipul ***Student*** pentru elementele din TreeSet. Clasa ***Student*** este definită în interiorul clasei ***Catalog***, implementează interfața ***Comparable*** și conține următoarele câmpuri: un ***String*** *name*, pentru reținerea numelui studentului, un ***double*** *media*, pentru medie, și un ***int*** *clazz* pentru reținerea numărului grupei din care face parte studentul. Compararea a doi studenți se realizează în funcție de medie (crescător), iar dacă acestea sunt egale, se realizează o sortare alfabetică. Se va defini si metoda toString!

Clasa trebuie să pună la dispoziție metode pentru următoarele operații: adăugarea unui student, căutarea unui student după nume (va returna null dacă nu există un student cu numele indicat în colecție), ștergerea unui student din catalog (primește ca parametru numele studentului), ordonarea alfabetică a studenților dintr-o grupă, primită ca parametru.

Pentru testare, se va completa instanțierea obiectului ***catalog2*** din interiorul clasei ***Task2***, pusă la dispoziție în arhiva laboratorului, astfel încât acest catalog să fie sortat descrescător după medie și la egalitate crescător după nume.

```
//Constructori
public Catalog (Comparator comparator ) ;
public Catalog ( ) ;

public void addStudent(String name, double media, int clazz);
public Student getStudent ( String name) ;
public void removeStudent ( String name) ;
public Catalog byClass (int clazz ) ;
// Contine o clasa anonima de tip Comparator ce va fi folosita la crearea Catalog-ului returnat
```

Problema 3

Definiți clasa **generică *LinkedList*** care va implementa interfața ***Iterable*** și va modela o listă liniară simplu înlănțuită: class LinkedList<T> implements Iterable<T>

În clasa ***LinkedList*** se vor defini:

- o clasă ***generica*** internă privată statică ***Node***

```
 private static class Node<T>
```

care conține două referințe: una pentru valoarea pe care o reține nodul și una pentru nodul următor din listă; și doi constructori: unul cu doi parametri (valoarea și nodul următor) și unul cu un parametru (valoarea) - va atribui valoarea null nodului următor;

- două elemente de tip ***Node***, reprezentând primul și respectiv ultimul element din listă;
- o clasă internă ***ListIterator*** ce va implementa interfața generica ***Iterator***;
- metodele prezentate în blocul de cod de mai jos.

```
//Insereaza un nod la inceputul listei
public void addFirst(T data);
//Insereaza un nod la sfarsitul listei
public void add(T data);
//Returneaza un obiect de tip ListIterator
public Iterator<T> iterator();
```

Pentru testare, se va folosi clasa ***Task3***, pusă la dispoziție în arhiva laboratorului.

Problema 4

Realizați un program care încearcă afișarea a unui tabel în două formate, astfel încât să fie ușor de importat într-un program de calcul tabelar (i.e. Excel - csv) sau pentru a fi ușor de citit pentru oameni - ASCII.

Definiți clasa ***Table*** pentru a modela un tabel care să poată reprezenta căsuțe de orice tip (de. ex, pe un rând puteți avea atât numere cât și șiruri de caractere). Clasa va conține un ***Vector*** de ***Vector*** de ***Object*** și clasele interne ***AsciiPrinter*** și ***CsvPrinter***, utilizate la afișarea în format ASCII și respectiv csv a unui obiect de tip ***Table***. Clasa ***AsciiPrinter*** va primi o listă de marimi ale fiecărei coloane pentru a putea să le aranjeze corespunzător, iar clasa ***CsvPrinter*** va primi un șir delimitator – în acest caz va fi ",".

Intuiți din clasa ***Task4*** care ar putea fi structura clasei ***Table*** și implementați-o.

Clasa ***AsciiPrinter*** va afișa tabelul sub următoarea formă:

```
#| Materie                         | An| Semestru| Credite|
1| Programarea calculatoarelor      | 1 | 1       | 6      |
2| Structuri de date                | 1 | 2       | 6      |
3| Programare Orientata pe Obiecte  | 2 | 1       | 6      |
```

Clasa ***CsvPrinter*** va afișa tabelul sub următoarea formă:

```
#,Materie,An,Semestru,Credite
1,Programarea calculatoarelor,1,1,6
2,Structuri de date,1,2,6
3,Programare Orientata pe Obiecte,2,1,6
```

```
public interface Printer, Table.print(Printer p)
System.out.format
```

poo/laboratoare/08.txt · Last modified: 2022/11/24 13:08 by carmen.odubasteanu